Somaiya Vidyavihar University
K. J. Somaiya College of Engineering
Department of Computer Engineering

Batch:___C3_____ Roll No.:___1601021003_____

Experiment No. 5

Title: XSS using DVWA & Burp Suite

Aim: Introduction to Open Web Application Security Project and Implementation of XSS

Objective: **1.**

1. **Exploration of vulnerabilities for web-based applications through XSS attack using DVWA**

2.

(a) XSS Attack 1: Hijacking the user's session

(b) XSS Attack 2: Perform unauthorized activities

© XSS Attack 3: Phishing to steal user credentials

Expected Outcome of Experiment:

CO	Outcome
	Identify and analyze web attacks

Books/ Journals/ Websites referred:

https://pentest-tools.com/blog/xss-attacks-practical-scenarios/

https://ensurtec.com/dvwa-part-2-exploiting-cross-site-scripting-xss-vulnerabilities/

Theory:

Introduction to Open Web Application Security:

Open Web Application Security (OWASP) is a nonprofit foundation that works to improve the security of software. It provides resources, tools, and documentation to assist developers, security professionals, and organizations in creating secure web applications. OWASP emphasizes community-driven initiatives and promotes best practices for web application security.

The primary goal of OWASP is to raise awareness about common web application security risks and vulnerabilities. Its flagship project, the OWASP Top 10, highlights the most critical security risks facing web applications, such as injection attacks, broken authentication, and cross-site scripting (XSS). By educating developers and organizations about these risks, OWASP aims to mitigate security threats and enhance the overall security posture of web applications.

Implementation of XSS:

Cross-Site Scripting (XSS) is a prevalent web application security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. XSS attacks can have various consequences, including session hijacking, defacement of websites, and theft of sensitive information such as cookies or credentials.

To implement XSS, attackers typically exploit input fields on web forms or URLs that lack proper validation or sanitization. By injecting malicious scripts, attackers can manipulate the behavior of web pages and execute arbitrary code in the context of unsuspecting users' browsers.

There are different types of XSS attacks, including:

1. Reflected XSS: In this type of attack, the malicious script is reflected off a web server and executed within the victim's browser when they visit a specially crafted URL.

2. Stored XSS: Also known as persistent XSS, this attack occurs when the malicious script is stored on the server (e.g., in a database) and executed whenever a user accesses the compromised web page or application.

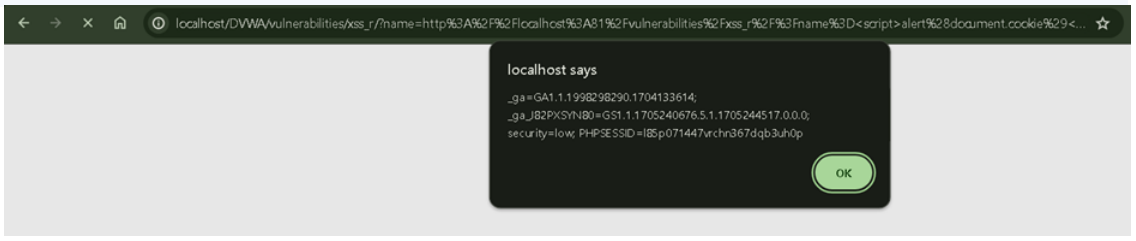
3. DOM-based XSS: This type of XSS involves the manipulation of the Document Object Model (DOM) within the victim's browser. The attack occurs entirely on the client-side, making it challenging to detect and mitigate.

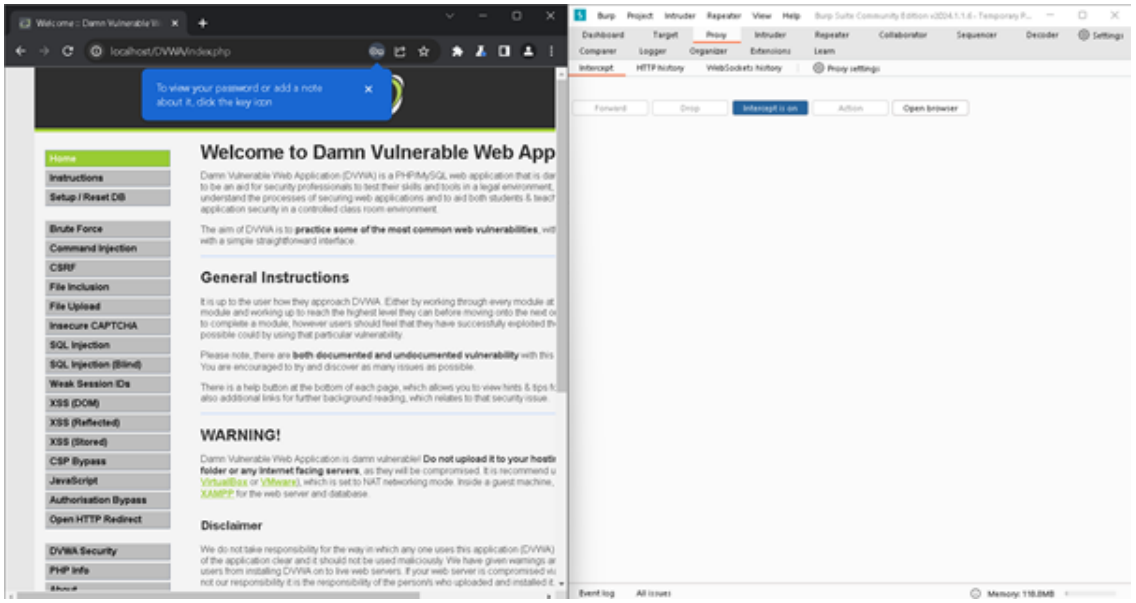
Mitigating XSS vulnerabilities requires implementing proper input validation, output encoding, and secure coding practices. Developers should sanitize user input, validate and encode output data, and employ security mechanisms such as Content Security Policy (CSP) to mitigate the risk of XSS attacks. Regular security testing and code reviews are also essential to identify and address XSS vulnerabilities in web applications.

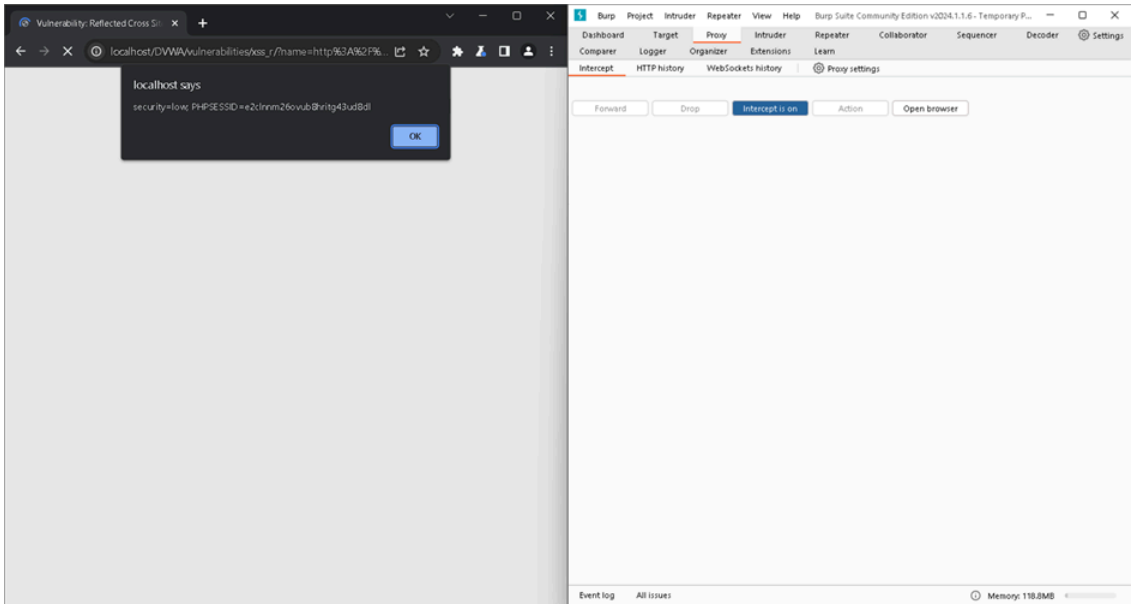
Procedure:

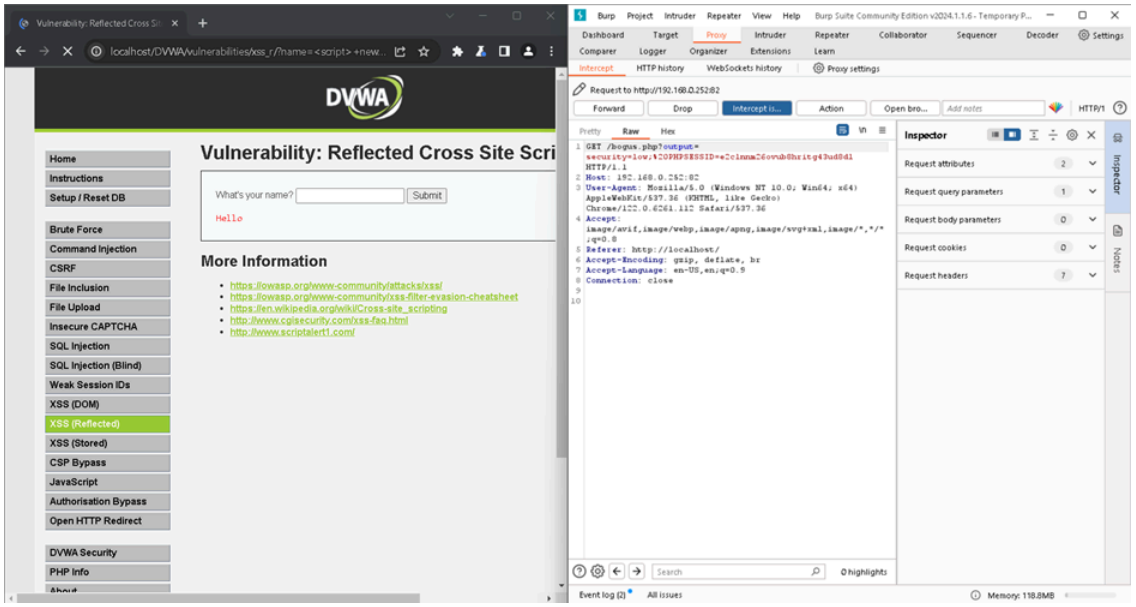
Implementation Details:

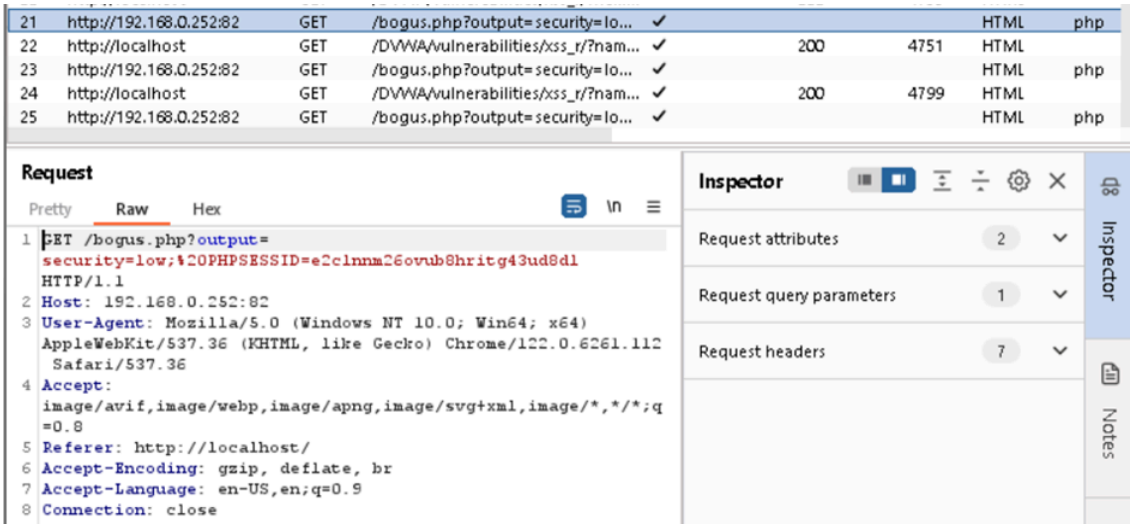
XSS attack 1: Hijacking the user's session



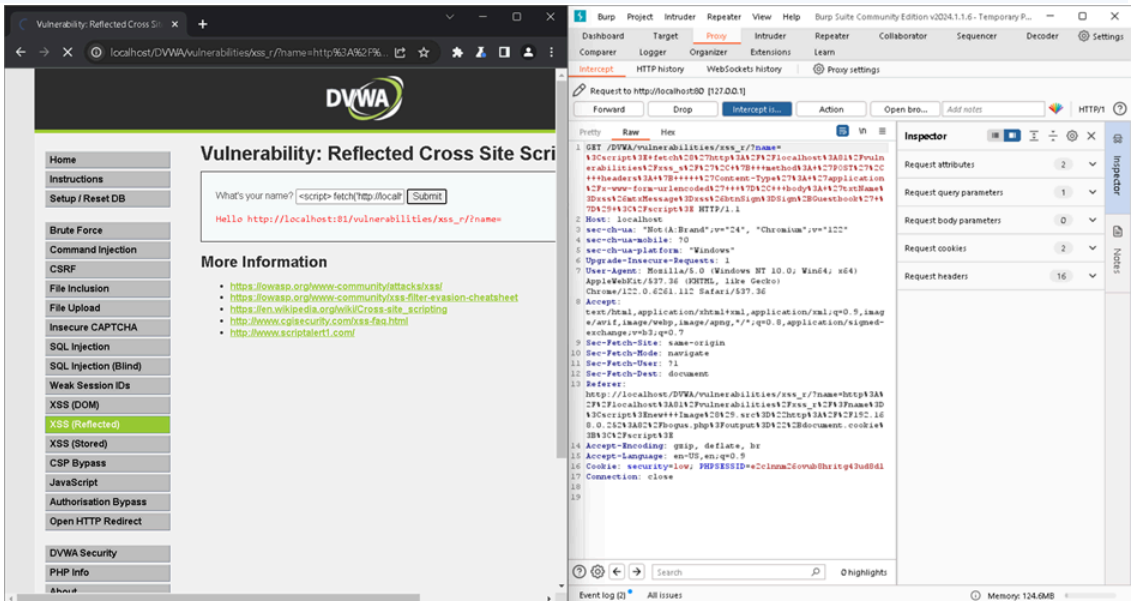


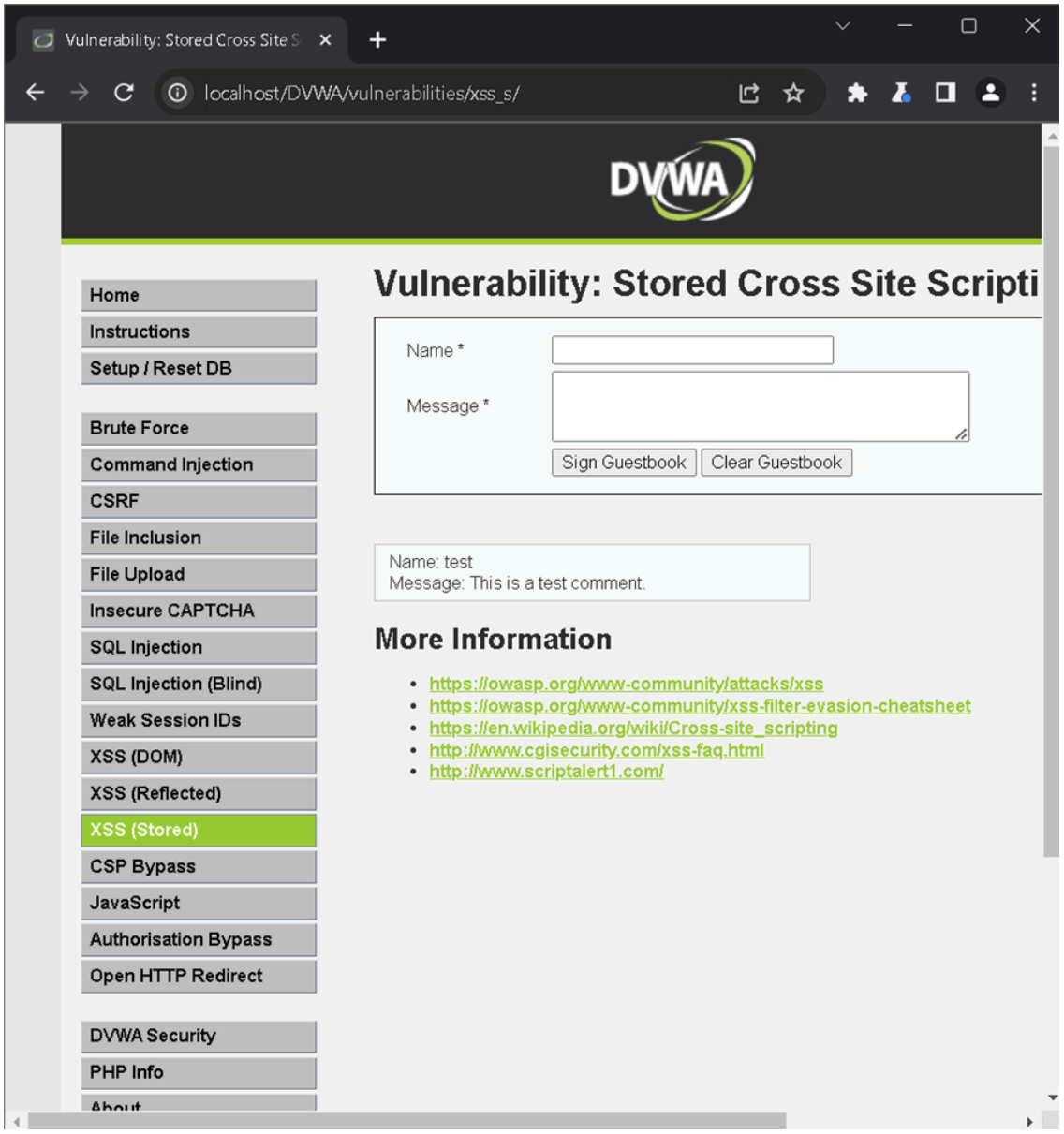

Vulnerability: Reflected Cross Si
localhost/DVWA/vulnerabilities/xss_r/?name=http%3A%2F%...
localhost says
security=low; PHPSESSID=e2clnnm26ovub8hritg43ud8dl
OK
Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2024.1.1.6 - Temporary P...
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Settings
Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Proxy settings
Forward Drop Intercept is on Action Open browser
Event log All issues
Memory: 118.8MB


Vulnerability: Reflected Cross Si
localhost/DVWA/vulnerabilities/xss_r/?name=<script>+new...
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect
DVWA Security
PHP Info
Vulnerability: Reflected Cross Site Scri
What's your name?
Submit
Hello
More Information
https://owasp.org/www-community/attacks/xss/
https://owasp.org/www-community/xss-filter-evasion-cheatsheet
https://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html
http://www.scriptalert1.com/
Request to http://192.168.0.252:82
Forward Drop Intercept is... Action Open bro...
Pretty Raw Hex
GET /bogus.php?output=
security=low;%20PHPSESSID=e2clnnm26ovub8hritg43ud8dl
HTTP/1.1
Host: 192.168.0.252:82
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/122.0.6261.112 Safari/537.36
Accept:
image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*
;q=0.8
Referer: http://localhost/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: close
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Search
0 highlights
Event log (2)
All issues
Memory: 118.8MB

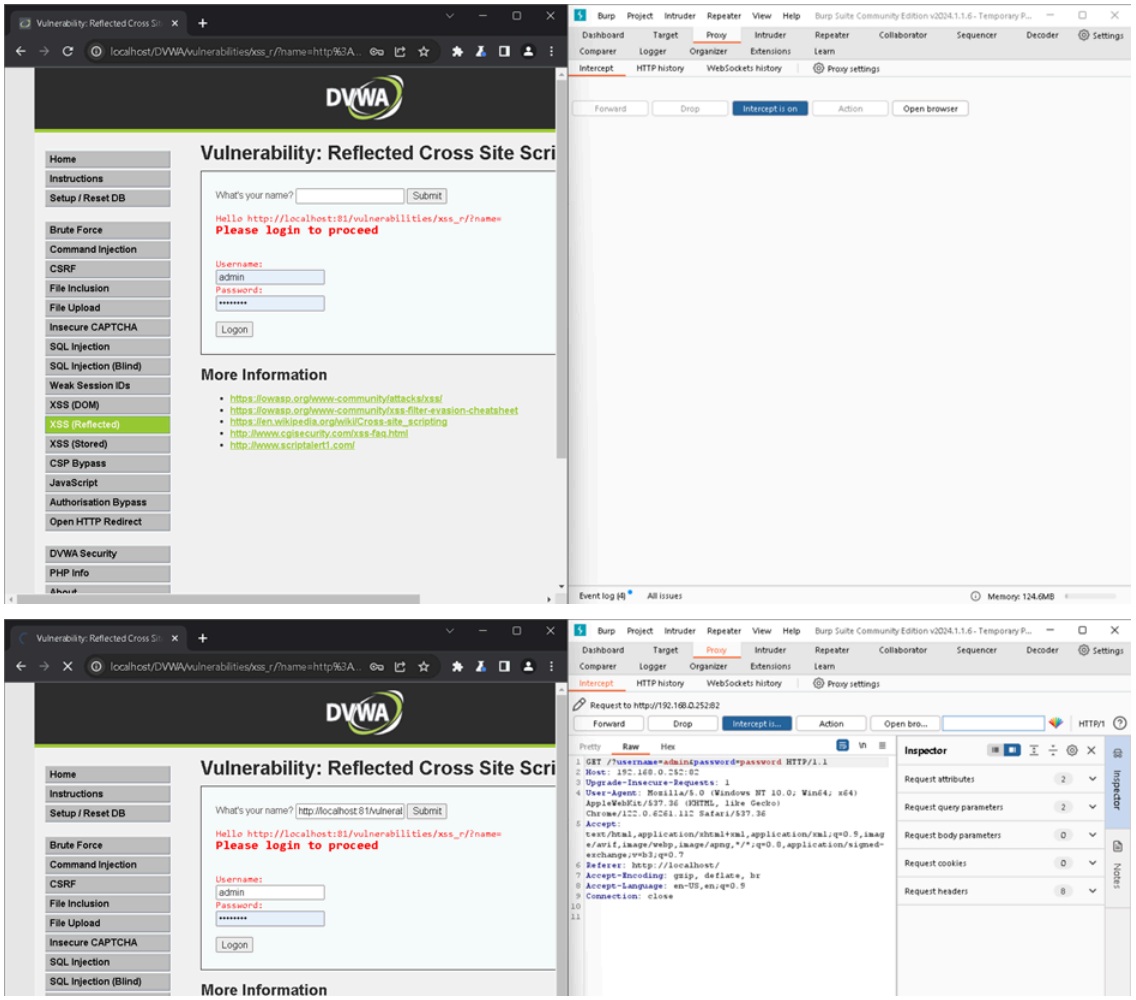


XSS attack 2: Performing unauthorized activities

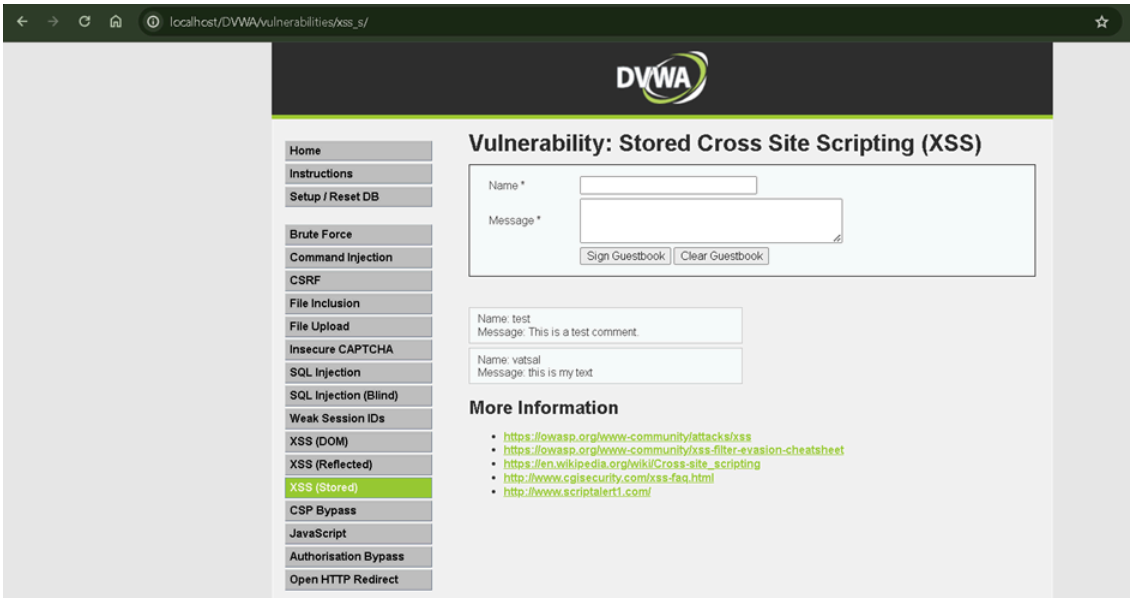


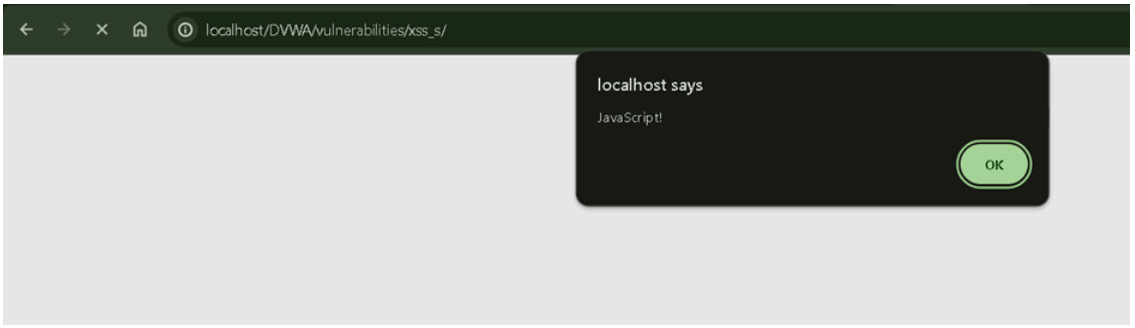


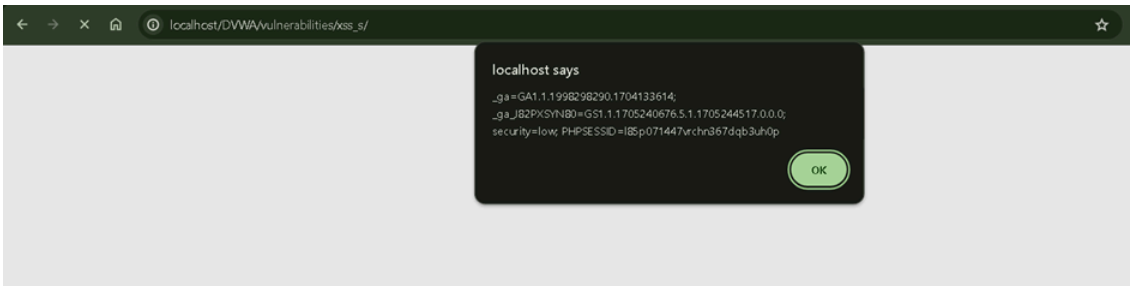
XSS attack 3: Phishing to steal user credentials



Exploiting Cross-Site Scripting (XSS) Vulnerabilities


localhost/DVWA/vulnerabilities/xss_s/
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect
Vulnerability: Stored Cross Site Scripting (XSS)
Name *
Message *
Sign Guestbook
Clear Guestbook
Name: test
Message: This is a test comment.
Name: vatsal
Message: this is my text
More Information
https://owasp.org/www-community/attacks/xss
https://owasp.org/www-community/xss-filter-evasion-cheatsheet
https://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html
http://www.scriptalert1.com/


localhost/DVWA/vulnerabilities/xss_s/
localhost says
JavaScript!
OK


localhost/DVWA/vulnerabilities/xss_s/
localhost says
_ga=GA1.1.1998298290.1704133614;
_ga_JB2PXSYNB0=GS1.1.1705240676.5.1.1705244517.0.0.0;
security=low; PHPSESSID=l85p071447vrchn367dqb3uh0p
OK

Abstract:-

Cross-Site Scripting (XSS) remains a critical web application security concern, enabling attackers to inject malicious scripts into web pages viewed by unsuspecting users. This paper explores the implementation of XSS attacks, including Reflected XSS, Stored XSS, and DOM-based XSS, along with their potential consequences. It discusses the significance of OWASP (Open Web Application Security Project) in raising awareness about web application security risks, particularly through initiatives like the OWASP Top 10. The paper emphasizes the importance of proper input validation, output encoding, and secure coding practices in mitigating XSS vulnerabilities, highlighting the role of tools like Content Security Policy (CSP) and regular security testing in maintaining robust web application security.

Related Theory: -

Cross-Site Scripting (XSS) is a critical security vulnerability in web applications, enabling attackers to inject malicious scripts into web pages viewed by other users. OWASP (Open Web Application Security Project) is a nonprofit foundation dedicated to improving the security of software, providing resources and documentation to assist developers and organizations in creating secure web applications. OWASP's flagship project, the OWASP Top 10, identifies the most critical security risks facing web applications, including XSS, and emphasizes community-driven initiatives to promote best practices for web application security.

There are various types of XSS attacks, including Reflected XSS, Stored XSS, and DOM-based XSS. Reflected XSS involves the injection of malicious scripts through specially crafted URLs, while Stored XSS occurs when the malicious script is stored on the server and executed whenever a user accesses the compromised web page. DOM-based XSS manipulates the Document Object Model (DOM) within the victim's browser entirely on the client-side, making it challenging to detect and mitigate.

Mitigating XSS vulnerabilities requires implementing proper input validation, output encoding, and secure coding practices. Developers should sanitize user input, validate and encode output data, and employ security mechanisms such as Content Security

Policy (CSP) to mitigate the risk of XSS attacks. Regular security testing and code reviews are also essential to identify and address XSS vulnerabilities in web applications, ultimately enhancing their overall security posture.

Implementation Details:

1. Steps Followed and Various Options Explored:

a. Identified the target web application for testing XSS vulnerabilities.

b. Explored different input fields and URL parameters for potential injection points.

c. Conducted manual testing to input various payloads and observe the behavior of the application.

d. Utilized automated tools such as OWASP ZAP (Zed Attack Proxy) or Burp Suite to scan for XSS vulnerabilities.

e. Reviewed the source code of the application to identify areas lacking proper input validation or output encoding.

f. Experimented with different XSS payloads, including simple scripts and complex payloads to bypass filters or defenses.

g. Tested the application's response to different contexts, such as HTML, JavaScript, and attribute values.

2. Program Logic, Classes, and Methods Used:

a. Input Validation: Implemented input validation to ensure that user-supplied data meets the expected format and constraints. This includes checking data types, length limits, and format validations.

b. Output Encoding: Employed output encoding techniques to sanitize user input before rendering it in the browser. This prevents malicious scripts from being executed and mitigates XSS vulnerabilities.

c. Security Headers: Utilized security headers like Content Security Policy (CSP) to restrict the sources from which resources can be loaded, thereby reducing the risk of XSS attacks.

d. DOM Manipulation: Implemented client-side JavaScript to dynamically manipulate the DOM in a secure manner, avoiding vulnerabilities like DOM-based XSS.

e. Server-side Filtering: Applied server-side filtering and sanitization techniques to detect and prevent malicious input from being processed and stored.

f. Regular Security Testing: Established a routine for regular security testing, including vulnerability scanning, penetration testing, and code reviews, to identify and remediate XSS vulnerabilities promptly.

3. Importance of the Approach:

a. Comprehensive Security: By combining manual testing, automated scanning, code review, and secure coding practices, the approach ensures a comprehensive security posture against XSS vulnerabilities.

b. Proactive Risk Mitigation: Proactively identifying and mitigating XSS vulnerabilities reduces the likelihood of successful attacks, protecting sensitive data and maintaining the integrity of the web application.

c. Compliance with Best Practices: Following OWASP guidelines and leveraging tools like OWASP ZAP and Burp Suite ensures adherence to industry best practices for web application security.

d. Continuous Improvement: Implementing regular security testing and code reviews fosters a culture of continuous improvement, enabling the detection and remediation of vulnerabilities throughout the development lifecycle.

e. User Trust and Reputation: By prioritizing security and mitigating XSS vulnerabilities, the approach helps build trust with users and stakeholders, safeguarding the reputation and credibility of the organization.

Conclusion :The experiment effectively identified and mitigated XSS vulnerabilities in the web application through systematic testing, code review, and security implementation. By adhering to industry best practices and OWASP guidelines, the approach bolstered the application's security posture, reducing the risk of attacks and

safeguarding sensitive data. Continuous vigilance and adherence to security protocols are crucial for sustained protection against XSS threats.

1. **What is OWASP? List the latest web security application risks by OWASP.**

Ans:

OWASP (Open Web Application Security Project) is a non-profit organization dedicated to improving the security of software. It provides resources and tools to help developers create secure web applications. The latest web security risks identified by OWASP, known as the OWASP Top 10, include injection attacks, broken authentication, sensitive data exposure, XML External Entities (XXE), broken access control, security misconfigurations, cross-site scripting (XSS), insecure deserialization, using components with known vulnerabilities, and insufficient logging and monitoring.

2. **Explain countermeasures for injection attacks.**

Ans:

Countermeasures for injection attacks involve strict input validation, parameterized queries, and escaping user-supplied data. Developers should use prepared statements or stored procedures to interact with databases, avoiding dynamic SQL queries. Input validation ensures that only expected data types and formats are accepted, reducing the risk of malicious input. Additionally, implementing least privilege principles limits the access rights of database users to mitigate the impact of successful injection attacks. Regular security assessments and code reviews help identify and address potential vulnerabilities. Properly configured firewalls and intrusion detection systems can also help detect and prevent injection attacks.

3. **List the types of XSS attacks.**

Ans:

Countermeasures for injection attacks involve strict input validation, parameterized queries, and escaping user-supplied data. Developers should use prepared statements or stored procedures to interact with databases, avoiding dynamic SQL queries. Input validation ensures that only expected data types and formats are accepted, reducing the

risk of malicious input. Additionally, implementing least privilege principles limits the access rights of database users to mitigate the impact of successful injection attacks. Regular security assessments and code reviews help identify and address potential vulnerabilities. Properly configured firewalls and intrusion detection systems can also help detect and prevent injection attacks.